FPGA BASED IMPLEMENTATION OF A FUZZY NEURAL NETWORK MODULAR ARCHITECTURE FOR EMBEDDED SYSTEMS

R. N. A. Prado[1]; J. D. Melo[2]; J. A. N. Oliveira[3]; A. D. Dória Neto[4];
Post Graduation Program on Electrical and Computational Engineering
Federal University of Rio Grande do Norte
Natal, Brasil
1- rnaprado@yahoo.com.br 2- jdmelo@dca.ufrn.br 3- nicolau@ufrnet.br 4-adriao@dca.ufrn.br

Abstract— **This paper presents a FPGA based approach for a modular architecture of Fuzzy Neural Networks (FNN) to embed with easily different topologies set up. The project is based on a Takagi – Hayashi (T-H) method for the construction and tuning of fuzzy rules, this is commonly referred as neural network driven fuzzy reasoning. The proposed architecture approach consists of two main configurable modules: a Multilayer Perceptron – MLP with sigmoidal activation function that composes the first module to determine a Fuzzy membership function; the second employs an MLP with pure linear activation function to define the consequents. The DSPBuilder® software along the Simulink® is used to connect, set and synthesize the Fuzzy Neural Network desired. Other hardware components employed in the architecture proposed cooperate to the system modularity. The system was tested and validated through a control problem and an interpolation problem. Several papers proposed different hardware architecture to implement hybrid systems by using Fuzzy logic and Neural Network. However, there is no approach with this specific neural network driven fuzzy reasoning by T-H method and the aim to be embedded. The Self-Organizing Map (SOM) and Levenberg-Marquardt backpropagation were used to train the FNN proposed off-line.**

Keywords- Fuzzy Neural Network, Takagi Hayashi method, neural network driven fuzzy reasoning, hybrid systems, DSPBuilder®, Modular Architecture, FPGA.

I. INTRODUCTION

Artificial intelligence (AI) methods are generally used to solve complex problems in engineering. Sometimes these methods are incapable to solve some problems individually, thus the methods can be combined to create hybrid systems and solve more complex problems.

Hybrid intelligent system generally involves two, three or more of these individual AI technologies that are either used in series or in an integrated way to produce advantageous results through synergistic interaction [7].

Fuzzy systems that have several inputs suffer from the curse of dimensionality. The Takagi – Hayashi (T-H) method is an automatic procedure to extract rules and can greatly reduce the number of rules in a high dimensional problem, making the problem tractable, thus [7].

Many papers proposed different solution for extracting rules and consequent parameters using neural network training methods, some papers described in literature are: 1) a probabilistic neural-fuzzy learning system for stochastic modeling [3] 2) fuzzy system adaptation using gradient-descent error minimization [16]; 3) fuzzy hierarchy error approach [2] 4) a self-organizing TS-type fuzzy network with support vector learning [4] 5) evolutionary learning of BMF fuzzy-neural networks using a reduced-form genetic algorithm [6] 6) optimization of a parameterized fuzzy system with symmetric triangular-shaped input membership functions and crisp outputs using gradient-descent error minimization [17] [18] [19]; 7) gradient-descent with exponential MFs [20]; and 8) gradient-descent with symmetric and non-symmetric LHS MFs varying connectives and RHS forms [21] [23].

Several papers proposed different FPGA architectures for Fuzzy Neural Networks (FNN) implementations, but none paper using neural network driven fuzzy reasoning with Takagi – Hayashi (T-H) method, some papers described in literature with similar intent are: 1) a type-2 self-organizing neural fuzzy system and its FPGA implementation [10]; 2) an experimental study on nonlinear function computation for neural/fuzzy hardware design [11]; 3) a hardware/software implementation of an adaptive neuro-fuzzy system [12]; 4) speedup of implementing fuzzy neural networks with high-dimensional inputs through parallel processing on graphic processing units [15]; 5) other applications only with fuzzy system like an FPGA-based online detection of multiple combined faults in induction motors through information entropy and fuzzy inference [13]; 6) a hardware implementation of an adaptive network-based fuzzy controller for dc–dc converter [22]; and 7) a fuzzy inference chip that allows for real-time online context switching [14].

This work presents a hardware approach for implementation of fuzzy neural network modular architecture for embedded systems. The project is based on a Takagi – Hayashi (T-H) method for the construction and tuning of fuzzy rules, this is commonly called neural network driven fuzzy reasoning. This method allows the construction of rules and membership functions through many different backpropagation training methods. The DSPBuilder® software along the Simulink® is used to connect, set and synthesize the desired Fuzzy Neural Network.

U.S. Government work not protected by U.S. copyright

The article is divided as follows: section two discusses the Fuzzy Neural Network (FNN) based on T-H method; section three deals with the FNN hardware project; section four shows details of FPGA structures for FNN implementation; section five discusses the method to set different FNN topology in this hardware architecture; section six shows the FPGA area analysis; section seven presents experimental results and discussions about the project and finally, section seven contains conclusions and future projects involving the issue under study.

II. FUZZY NEURAL NETWORK

The fuzzy systems based on parametric equations consist of a set of membership functions set by equations, instead of rules and a set of consequences being also defined by parametric equations. Both equations depend on the input values of the system. The complexity of the fuzzy systems based on parametric equations is in its own definition of free variables of these parametric equations, which can increase significantly depending on the amount of membership functions and consequences of the system [5] [7].

The Fuzzy Neural Network – FNN, in Takagi-Hayashi method, has aimed to set numerous parametrical membership functions and consequents from the Multilayer Perceptron (MLP) Neural Networks learning (backpropagation) with fuzzy parametric controller systems. This feature simplifies the complexity of the increase of parametric fuzzy systems when they have many equations to be set therefore the backpropagation training is used to define these parametric equations.

III. STRUCTURE OF FUZZY NEURAL NETWORK - FNN HARDWARE

In a Fuzzy Neural Network - FNN of Takagi Hayashi method (T-H) there are two types of neural network Multilayer Perceptron - MLP: the first is a neural structure for the arrangement of the FNN membership functions, a network with one hidden layer with tangent sigmoid activation function in neurons and the output layer with tangent sigmoid activation function, we call this structure "SIG MLP" in this paper, the second is a neural structure to determine the consequent parameters of an FNN, which consists of a neural network with one hidden layer with tangent sigmoid activation function in neurons and the output layer with purelin activation function, we call this structure "LINEAR MLP" in this paper [5] [7]. Both structures process data in sixteen-bit fixed point representation.

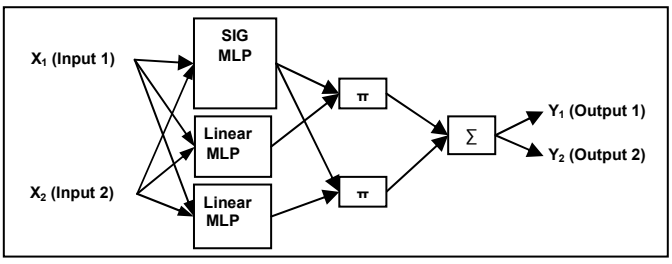

Figure 1. FNN T-H hardware architecture.

Figure 1 shows an FNN T-H with two inputs and two outputs. This requires one SIG MLP structure for the arrangement of membership functions and two LINEAR MLP structures to determine the consequents of clusterized data. For the arrangement of the membership functions, it is necessary a collection of multipliers neurons, represented by the symbol "π". They are used to make the weighting between membership functions and the consequents.

IV. FPGA IMPLEMENTATION OF FNN T-H STRUCTURES

A. Single Neuron Implementation

The neuron proposed by McCulloch and Pitts was used as a reference, and the neuron architecture proposed in this study followed the implementation model created in VHDL by [1], as described in Register Transfer Level Design (RTL Design) shown in Figure 2. The neuron was divided into two functional blocks. The first is a linear combiner, responsible for summing weighted synaptic inputs and the second is responsible for calculating the activation function, denoted blocks NET and FNET respectively.

The proposed VHDL Neuron processes numerical data only in fixed point arithmetic, determining the amount of bits for representativeness of the number that is 16 bits with signal. The 16 bits fixed point representation provides an error of "0,448 x 10^{-6}" for this system. This error value doesn't affect perceptibly the simulations in the section VII.

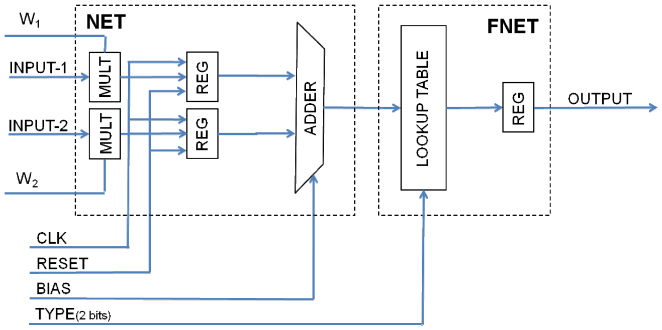

Figure 2. Basic Neuron described in Register Transfer Level

Implementation of the sigmoid or sigmoid tangent activation function in FPGA is achieved by using a lookup table, whose structure is composed by a comparator block and two parallel ROMs with 16 x 21 bits of data. The reason for choosing a lookup table to simulate the sigmoid tangent function is related to the cost and difficulty of implementing it in FPGA in any other way. Since this is a discretization, an error in the values obtained as a response can be clearly seen. To compensate them and consider them in the program developed, the activation function applied would not be a mathematical function, but rather a table with 21 points, in which values are previously defined [8].

To determine these values, the solution adopted represents the function by means of a set of linearly interpolated points, so that the difference between the function curve and the interpolated points curve is minimal. Thus, a computational intelligence technique known as a genetic algorithm was used where each individual represents a set of different points from

the interpolator, thereby minimizing the mean square error. After the execution of the genetic algorithm, 21 points from the table are obtained.

B. SIG MLP Implementation

For the development of FNN hardware, SIG MLP structure, artificial neurons described in VHDL were used and follow the model proposed in [1]. An output controller block was used to control the data flow and maintain the system parallelism and synchronism, in order to indicate when the SIG MLP structure processed the input data given. In FNN, the SIG MLP network is responsible for the arrangement of the membership functions. Below, Figure 3 shows the structure of the SIG MLP.

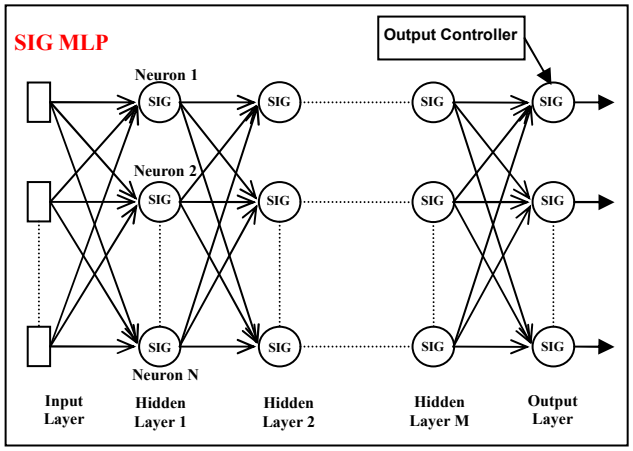

Figure 3. SIG MLP hardware structure.

C. MLP PURE Implementation

The LINEAR MLP structure uses artificial neurons described in VHDL and follows the model proposed by the SIG MLP structure, but without the use of activation function in the output layer in order to save silicon area on the FPGA and considering that LINEAR MLP structure uses the linear pure activation function in output layer. An output controller block was used to control the data flow and maintain the system parallelism and synchronism, in order to indicate when the LINEAR MLP structure processed the input data given. Next, Figure 4 shows the structure of the LINEAR MLP.

The logical operator "AND" was used in multiplier nodes structure (π in Figure 1). The calc made by π operator is performed by a T-NORM [5]. A tasks flag block was used to control the data flow and maintain the system parallelism and synchronism.

A small state machine is used to control the processing of data and keep the synchronization among the blocks that performs a new set of input data with only three states to complete a processing cycle and a starting state runs only when starting the system. After the initialization state, the first cycle state sets up the inputs, in the second state the simultaneous processes occur among SIG MLP and LINEAR MLP networks. In the third state, the multipliers synchronize operations to be transferred to the outputs. The initialization state defines the initial set up of all components of the architecture.

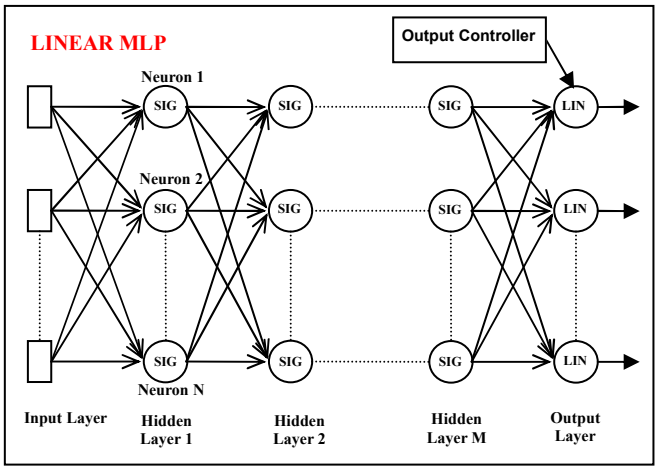

Figure 4. LINEAR MLP hardware structure.

V. SET DIFERENT FNN TOPOLOGY IN THIS HARDWARE ARCHITECTURE

The DSP Builder® allows the setting of a desired topology of FNN. The SIG MLP, LINEAR MLP and multiplier nodes structures were previously described in VHDL and tested individually. In DSP Builder®, you can define a particular configuration of hardware through flowchart diagrams of the Toolbox Simulink®, Matlab®, and it allows the synthesizing in FPGA. These functional blocks can be developed in VHDL components or even common components to Simulink®.

Using Simulink®, the architecture of FNN will be created with the insertion and interconnection of those functional blocks: LINEAR MLP; SIG MLP; multiplier blocks and output adders. The amount of components depends on the desired FNN architecture. The connections between the blocks in each structure are made by connecting the various blocks via the flowchart diagram in Simulink®. Different architectures of LINEAR MLP or SIG MLP can be created in DSPBuilder® with different amount of rules and, consequently, amount of neurons.

Figure 5 denotes the assembly and the interconnection of functional blocks and SIG MLP and LINEAR MLP described in VHDL using Simulink® diagrams. The controller is not being visualized in the figure, because the image would look with excess connections, making viewing difficult. Note that we used single input registers and four output connections to replicate the output of the SIG MLP structure.

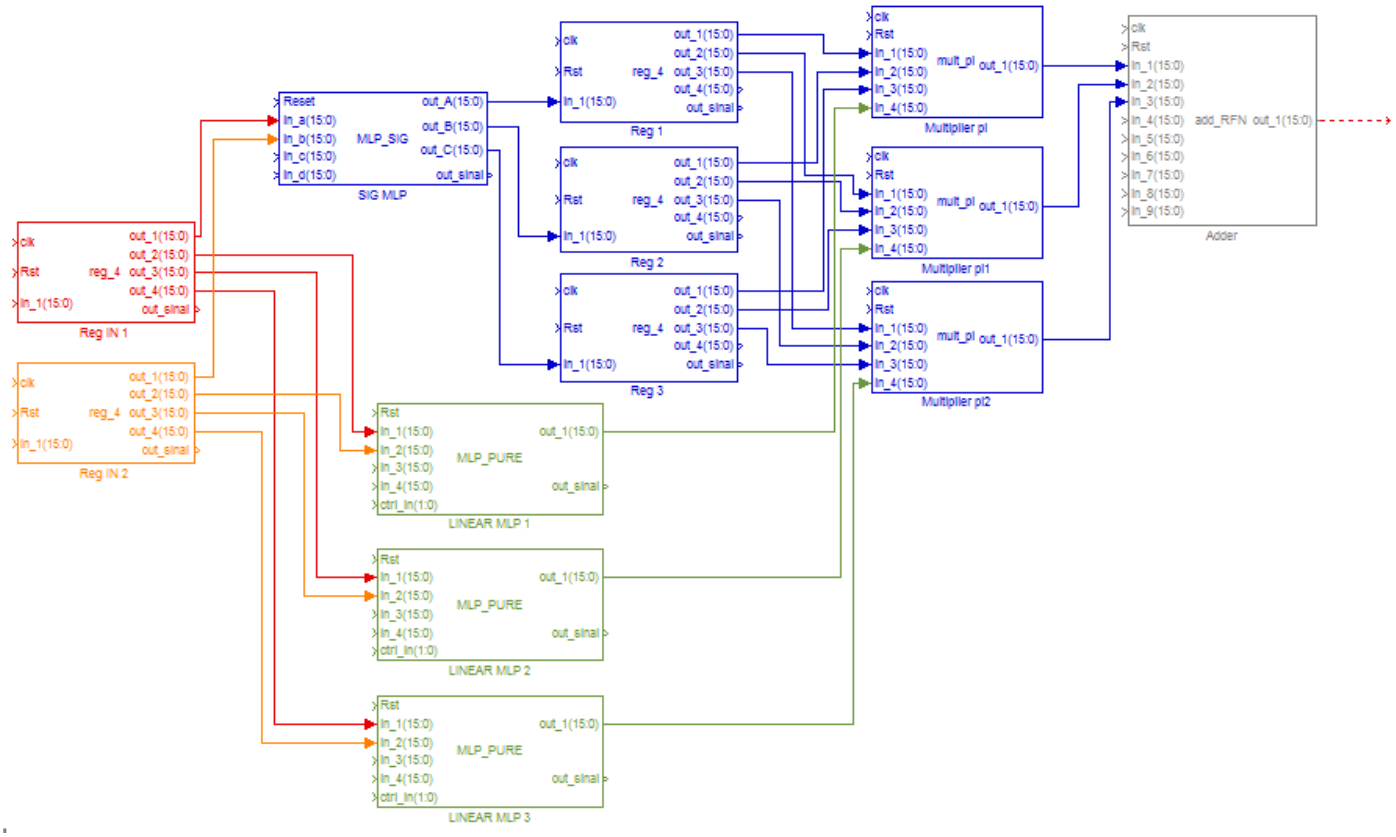

Figure 5. FNN T-H connections on DSPBuilder diagram.

In Figure 5, the FNN architecture is shown with two inputs and one output. To perform this architecture, it is necessary to use three LINEAR MLP blocks and one SIG MLP block along with three sets of π multiplier nodes blocks connected to the outputs of MLP blocks, and one adder block in the output of π block. The red block is the first input register and the orange block is the second input register.

VI. FPGA Area and Cycle Analisys in the Proposed Architecture

In this architecture, the number of logic elements used varies according to the topology of the desired FNN. The neurons number in FNN defines the amount of functional blocks that must be inserted. The reduced amount of FPGA area used in embedded systems is important to obtain energy-efficient chips and inexpensive devices. The number of logic elements used in two FNN- T-H networks was analyzed by using the software Altera's Quartus ® II which is shown in Table I below.

As observed in the controller of the proposed system, few cycles occur (execution of each layer) to generate the results of one process of a specific FNN T-H architecture. These cycles are used to enable the inputs, processing the MLP networks, synchronize MLP outputs data and calculate the last operations (t-norm and addition) to obtain the resulting control signal. The cycle analysis and maximum frequency reached by the proposed system was analyzed by using the software Altera's Quartus® II. Table II describes the number of cycles and maximum frequency related to the size of FNN T-H network topology used.

TABLE I. FPGA Area Analisys

Functional Block	**Device EP2C35F672C6 (Cyclone II)**
	Logical Elements / FPGA Area (Total use)
Linear Neurons	355 / 33216 (<1%)
Sigmoidal Neurons	560 / 33216 (<2%)
π Block	69 / 33216 (<1%)
Network 1 (problem 1)	25935/ 33216 (78.02 %)
Network 2 (problem 2)	10035 / 33216 (30,21%)

TABLE II. Cycle Analysis

Situation	**Tested FNN T-H Networks**			
	Network 1		***Network 2***	
MLP type	**1 SIG MLP**	**3 Linear MLP**	**1 SIG MLP**	**2 Linear MLP**
FNN T-H topology input/hidden1/output	4 /10 /3	4 / 7 / 1	1 / 5 / 2	1 / 5 / 1
Number of Cycles / Maximum frequency of the network input / Mean number of blocks used. **Proposed architecture.**	4 Cycles / 15,81 MHz / 25935		4 Cycles / 15,81 MHz / 10035	

VII. EXPERIMENTAL RESULTS

The FNN T-H architecture proposed was tested in a real industrial problem, a control of two-tanks system in a simulation case. The simulation problem confirms the capability of the proposed system to be used in a real embedded system case. The system model is shown in Figure 6.

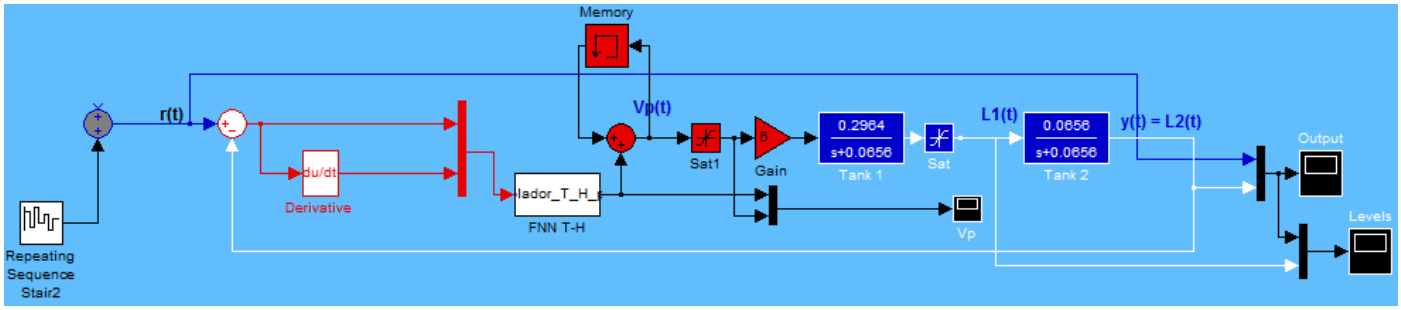

Figure 6. System Diagram on Simulink.

The T-H method also implements methods for reducing the neural network inputs to a small set of significant inputs and checking them for overfitting during training [7]. The four steps to train the FNN T-H network are:

Step 1: The training data x is clustered into r groups: R_1, R_2,..., $R_s\{s=1,2,...,r\}$ with nts terms in each group. Note that the number of inferencing rules will be equal to r.

Step 2: The NNmem neural network is trained with the targets values selected as:

$$w_i^s = \begin{cases} 1, & x_i \in R^s \\ 0, & x_i \notin R^s \end{cases} \quad i = 1,...,n_t^s; \quad s = 1,...,r \tag{1}$$

The outputs of NNmem for an input xi are labeled wis, and are the membership values of xi to each antecedent set Rs.

Step 3: The NNs networks are trained to identify the consequent part of the rules. The inputs are {xi1s,...xims}, and the outputs are yi=1,2,...,nt.

Step 4: The final output value y is calculated with a weighted sum of the NNs outputs.:

$$y_i = \frac{\sum_{s=1}^{r} \mu_{A^s}(x_i) \cdot \{u_s(x_i)\}_{\inf}}{\sum_{s=1}^{r} \mu_{A^s}(x_i)} \quad i = 1,2,...n \tag{2}$$

where $u_s(x_i)$ is the calculated output of NNs.

The train target used in all layers during the training was extracted from a Fuzzy controller developed by [9]. The two inputs visualized in FNN T-H block in Figure 6 corresponding to the setpoint error and the derivative setpoint error. The Self-Organizing Map was used to obtain the clustered data for training all networks (SIG MLP and LINEAR MLP). This clusterization is necessary in T-H training method described in Step 1. The data was clustered in three groups, thus, three LINEAR MLP networks were necessary. Each LINEAR MLP networks were trained with one group of clustered data. In the three groups clustered of 24.000 data, the first group obtains 50,23% of data, the second group has 41,61% and the third group has 8,16% of data.

The topology of FNN T-H used to control the two-tanks system is: three LINEAR MLP networks with 7 neurons on hidden layer and one neuron in output layer; one SIG MLP with 10 neurons on hidden layer and three neurons in output layer. Some authors suggest using one SIG MLP to create the membership functions of each input [5].

The simulation results of two-tanks system control are shown in Figure 7. The intent of the controller used as a model to train the FNN T-H network is to obtain a soft approximation of the setpoint searched. The aim to embed an FNN T-H on a chip to use in a real industrial problem was reached in simulation tests. The next step is to try the system in a real two-tank problem.

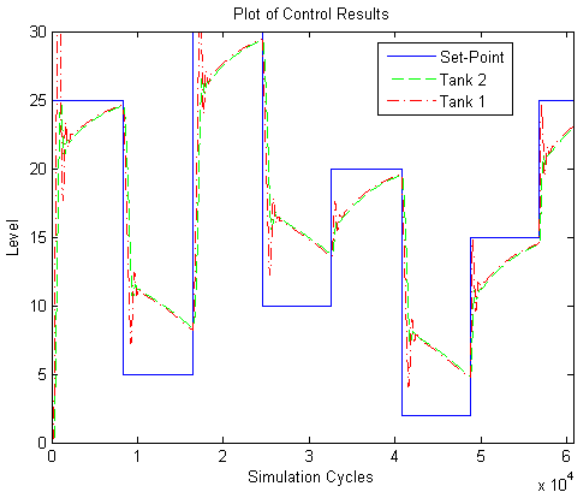

Figure 7. Results of two-tanks system control.

Other test with the FNN T-H hardware architecture proposed is the interpolation problem of the Sinc function. The topology of FNN T-H used to interpolation of Sinc function is: two LINEAR MLP networks with 5 neurons on the hidden layer and one neuron in the output layer; one SIG MLP with 5 neurons on the hidden layer and two neurons in the output layer. The results comparison is shown in Figure 8.

The result denotes a minimal difference between the original function and the network interpolation.

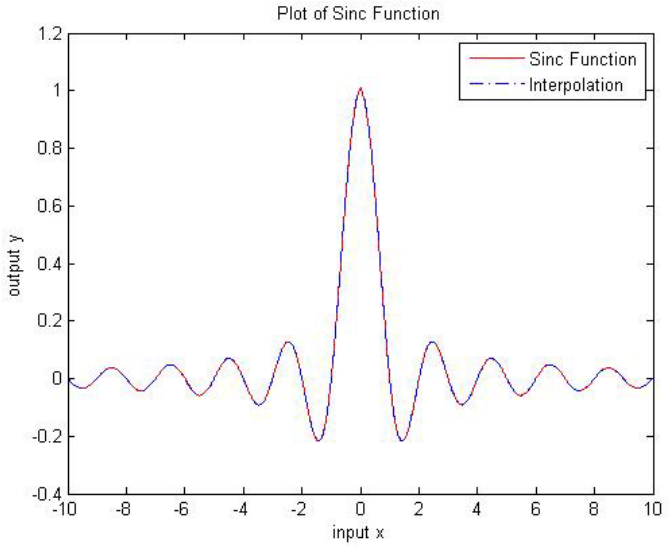

Figure 8. Results of Sinc fuction interpolation.

VIII. CONCLUSIONS

The paper proposes an FPGA based modular architecture of Fuzzy Neural Networks in Takagi Hayashi method (FNN T-H) for embedding, the architecture was implemented successfully. With this proposed system is possible to embed the FNN T-H method in a chip to solve many problems with simple topology configuration. In one of the secondary objectives, we were able to ease the modularization of the architecture to be used in different FNN T-H topologies from the block diagram. The DSPBuilder® was effective in providing an environment to configure the network and facilitate the synthesis of the project to be embedded.

The hardware architecture proposed allows FNN T-H on FPGA devices to be embedded and be used for applications in the control area, prediction problems, interpolation and other problems. The two-tank system control validated the implementation and showed reliable results. The MLP networks provide a soft computation to generate the parametric equations of membership functions and consequents equations.

The training method proved to be efficient, including the Self-Organizing Map (SOM) for data clustering and Levenberg-Marquardt backrpopagation to train all MLP networks (SIG MLP and LINEAR MLP).

In a second step, the proposed architecture will be used to develop a system with online training to embed control problems using FNN T-H. The main intention is to update the weights if the system has a complex dynamic that changes over time.

REFERENCES

[1] R. N. A. Prado, J. A. N. Oliveira, A. Doria Neto, J. D. Melo and C. A. A. Silva. "FPGA Based Architecture For Easy Configuration Of Multilayer Perceptron Neural Network Topologies". X Brazilian Congress in Computational Inteligence – X CBIC, Fortaleza, November, 2011.

[2] A. Wu and P. K. S. Tam. "A Fuzzy Neural Network Based on Fuzzy Hierarchy Error Approach". IEEE Transactions On Fuzzy Systems, Vol. 8, No. 6, December 2000.

[3] H. X. Li and Z. Liu. "A Probabilistic Neural-Fuzzy Learning System for Stochastic Modeling". IEEE Transactions On Fuzzy Systems, Vol. 16, No. 4, August 2008.

[4] C. F. Juang, S. H. Chiu and S. W. Chang. "A Self-Organizing TS-Type Fuzzy Network With Support Vector Learning and its Application to Classification Problems". IEEE Transactions On Fuzzy Systems, Vol. 15, No. 5, October 2007.

[5] SIMÕES, M. G. SHAW, I. S. "Controle e Modelagem Fuzzy". 2. Ed. Rev. e Amp. Local: Editora Blucher, FAPESP, 2007.

[6] W. Y. Wang and Y. H. Li. "Evolutionary Learning of BMF Fuzzy-Neural Networks Using a Reduced-Form Genetic Algorithm". IEEE Transactions On Systems, Man, And Cybernetics-Part B: Cybernetics, Vol. 33, No. 6, December 2003.

[7] L. H. Tsoukalas and R. E. Uhrig. "Fuzzy and Neurak Approaches in Engineering". Wiley Interscience Publications. 1997.

[8] D. R. C. Silva, A. D. Doria Neto, L. A. Guedes, J. D. Melo. "Neural Networks Implementation in Foundation Fieldbus Environment: A Case Study in Neural Control". International Journal of Factory Automation, Robotics and Soft Computing, v. 3, p. 48-54, 2006.

[9] A. M. P. P. Filho, K. R. Lopes, V. L. C. M. da Silva, D. L. Martins, A. D. Dória Neto, J. D. de Melo ; L. A. H. G. Oliveira . Controle de uma Coluna Debutanizadora Simulada Utilizando um Controlador Fuzzy Embarcado em uma Rede Foundation Fieldbus. In: IX SBAI - Simpósio Brasileiro de Automação Inteligente, Brasília-DF, 2009.

[10] C. F. Juang and Y. W. Tsao. "A Type-2 Self-Organizing Neural Fuzzy System and Its FPGA Implementation". IEEE Transactions On Systems, Man, And Cybernetics—Part B: Cybernetics, Vol. 38, Nº. 6, December 2008.

[11] K. Basterretxea, J. M. Tarela, I. del Campo and G. Bosque. "An Experimental Study on Nonlinear Function Computation for Neural/Fuzzy Hardware Design". IEEE Transactions On Neural Networks, VOL. 18, NO. 1, January 2007.

[12] I. del Campo, J. Echanobe, G. Bosque and J. M. Tarela. "Efficient Hardware/Software Implementation of an Adaptive Neuro-Fuzzy System". IEEE Transactions On Fuzzy Systems, Vol. 16, Nº. 3, June 2008.

[13] R. J. Romero-Troncoso, R. Saucedo-Gallaga, E. Cabal-Yepez, A. Garcia-Perez, R. A. Osornio-Rios, R. Alvarez-Salas, H. Miranda-Vidales and N. Huber. "FPGA-Based Online Detection of Multiple Combined Faults in Induction Motors Through Information Entropy and Fuzzy Inference". IEEE Transactions On Industrial Electronics, Vol. 58, Nº. 11, November 2011.

[14] Q. Cao, M. H. Lim, J. H. Li, Y. S. Ong and W. L. Ng. "A Context Switchable Fuzzy Inference Chip" IEEE Transactions On Fuzzy Systems, Vol. 14, Nº. 4, August 2006.

[15] C. F. Juang, T. C. Chen and W. Y. Cheng. "Speedup of Implementing Fuzzy Neural Networks With High-Dimensional Inputs Through Parallel Processing on Graphic Processing Units". IEEE Transactions On Fuzzy Systems, Vol. 19, Nº. 4, August 2011.

[16] I. Hayashi, H. Nomura, H. Yamasaki and N. Wakami. "Contruction of Inference Rules by NDF and NDFL". International Journing of Approximating Reasoning , Vol 6, pp 241-266, 1992.

[17] H. Nomura, I. Hayashi and N. Wakami. "A Self-tunning method of Fuzzy Reasoning by Genetic Algorithms". In Fuzzy Control Systems, A Candel and G. Langholz, eds., CRC Press, Boca Raton, pp. 338-354, FL 1994.

[18] J. S. Jang and C. T. Sun. "Neuro-Fuzzy Modeling and Control". Proceedings of the IEEE, Vol 83, Nº3, pp. 378-406, March 1995.

[19] L. X. Whang. "Adaptative Fuzzy Sisystem and Control". Prentice Hall, Englewood Clifs, NJ, 1994.

[20] H. Hichihashi, T. Miyoshi and K. Nagasaka. "Computed Tomography by Neuro-Fuzzy Inversion". Proceedings 1993 Joint Conference on

Neural Networks, Part 1 (of 3), Nagoya, pp. 709-712, October 25-29, 1993.

[21] F. Guély and P. Siarry. “Gradient Descent Method for Optimization Various Fuzzy Rules”. In Proceedings of Second IEEE International Conference on Fuzzy Systems. Pp 1241-1246, San Francisco 1993.

[22] A. Rubaai, A. R. Ofoli, L. Burge III and M. Garuba. “Hardware Implementation of an Adaptive Network-Based Fuzzy Controller for DC–DC Converters”. IEEE Transactions On Industry Applications, Vol. 41, Nº. 6, November/December 2005.

[23] J. F. Hurdle. “The Synthesis of Compact Fuzzy Neural Circuits” IEEE Transactions On Fuzzy Systems, Vol. 5, Nº. 1, February, 1997.